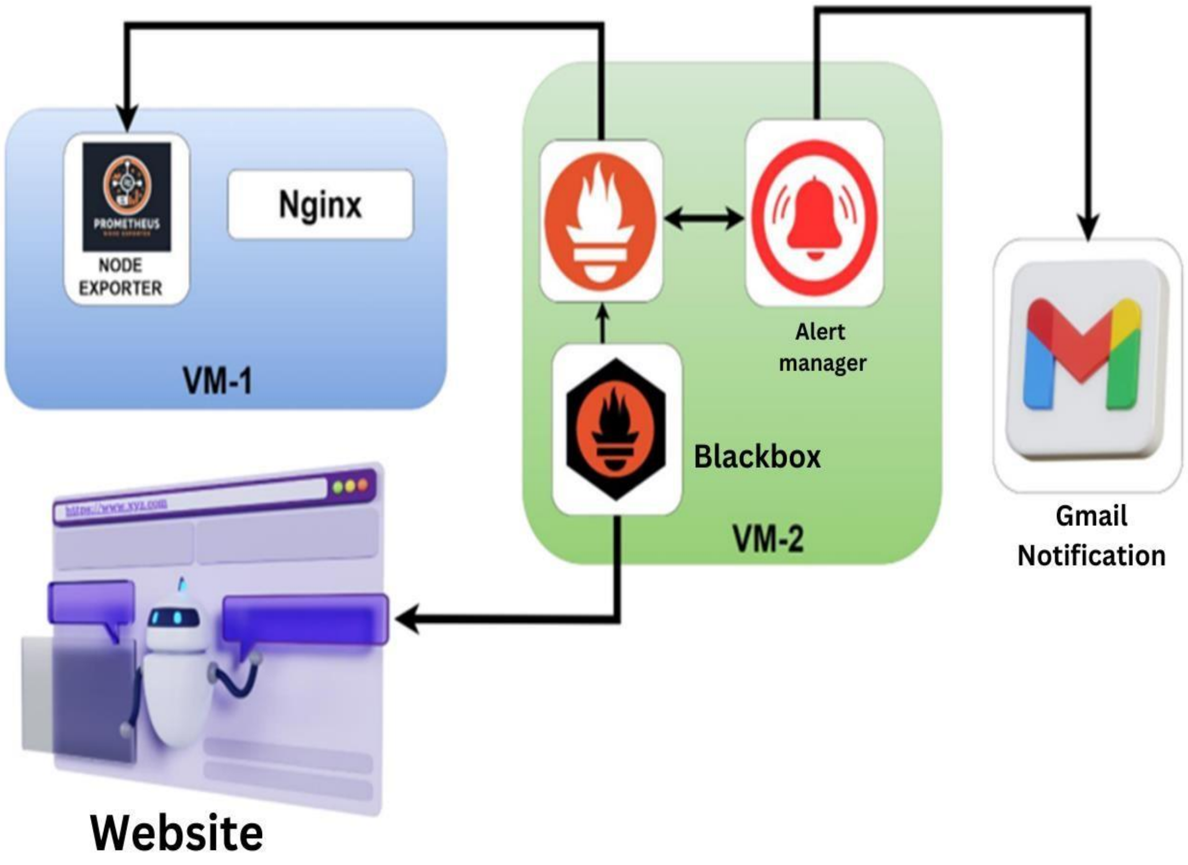
DevOps Monitoring Project

Introduction

In this project, we implemented a comprehensive monitoring solution using Prometheus and various exporters to ensure the reliability and performance of a web application hosted on AWS EC2 instances. This setup includes Node Exporter for hardware and OS metrics, Blackbox Exporter for probing endpoints, and Alertmanager for handling alerts. Gmail integration was also configured to receive notifications for critical alerts.

Architecture



Project Components and Best Practices

Components

1. **EC2 Instances**:

- **Instance 1**: Hosts the web application, Node Exporter, and Nginx.
- **Instance 2**: Hosts Prometheus, Blackbox Exporter, and Alertmanager.

2. **Prometheus**: Centralized monitoring tool for collecting and querying metrics.
3. **Node Exporter**: Collects hardware and OS-level metrics from the web server.
4. **Blackbox Exporter**: Probes endpoints to monitor uptime and response time.
5. **Alertmanager**: Manages alerts sent by Prometheus based on defined rules.
6. **Gmail Integration**: Sends email notifications for critical alerts.

Alerts

1. **InstanceDown:** Triggers when an instance is down for more than 1 minute.
2. **WebsiteDown**: Triggers when the website is down for more than 1 minute.
3. **HostOutOfMemory:** Triggers when available memory is less than 25% for more than 5 minutes.
4. **HostOutOfDiskSpace**: Triggers when the root filesystem has less than 50% available space.
5. **HostHighCpuLoad:** Triggers when CPU load is above 80% for more than 5 minutes.
6. **ServiceUnavailable:** Triggers when the node exporter service is unavailable for more than 2 minutes.
7. **HighMemoryUsage**: Triggers when memory usage exceeds 90% for more than 10 minutes.

8. **FileSystemFull:** Triggers when any filesystem has less than 10% available space for more than 5 minutes.

Best Practices

1. **Define Clear Objectives and Metrics**

- Identify Key Metrics: Determine which metrics are critical for the health and performance of your application and infrastructure (e.g., CPU usage, memory usage, response times, error rates).
- Set Baselines and Thresholds: Establish baseline performance levels and set thresholds for alerts to distinguish between normal and abnormal behavior.

2. **Use Multiple Data Sources**

- Combine Metrics and Logs: Use both metrics and logs to get a comprehensive view of the system's health.
- Integrate Various Exporters: Use relevant exporters (Node Exporter, Blackbox Exporter, etc.) to collect metrics from different parts of your infrastructure.

3. **Implement Robust Alerting**

- Define Relevant Alerting Rules: Create alerting rules that cover various failure scenarios and performance degradation.
- Avoid Alert Fatigue: Ensure alerts are actionable and avoid too many false positives. Group related alerts to reduce noise.
- Use Multiple Notification Channels: Configure alerts to be sent via multiple channels (email, SMS, chat tools) to ensure they are noticed.

4. **Ensure High Availability and Redundancy**

- Deploy Across Multiple Regions: Set up monitoring components in multiple regions to avoid single points of failure.
- Backup and Replicate Data: Regularly back up Prometheus data and configuration files. Use replication to ensure data availability.

5. **Optimize Performance and Resource Usage**

- Tune Scrape Intervals and Retention Policies: Set appropriate scrape intervals and data retention policies to balance between data granularity and resource usage.

Phase 1: Initial Setup

Provision EC2 Instances:

○ Launch two EC2 instances in your preferred region. ○ Configure security groups to allow necessary traffic (SSH, HTTP, Prometheus, etc.).

Launch Virtual Machine using AWS EC2

Here is a detailed list of the basic requirements and setup for the EC2 instance i have used for running Jenkins, including the specifics of the instance type, AMI, and security groups.

EC2 Instance Requirements and Setup:

1. Instance Type

- Instance Type: `t2.medium`

- vCPUs: 2

- Memory: 8 GB

- Network Performance: Moderate

2. Amazon Machine Image (AMI)

- AMI: Ubuntu Server 24.04 LTS

3. Security Groups

Security groups act as a virtual firewall for your instance to control inbound and outbound traffic.

Adjust security settings to allow necessary ports:

- Prometheus: 9090
- Alert manager: 9093
- Blackbox Exporter: 9115
- Node Exporter: 9100
- Email transmissions: 587

Phase 2: Install and Configure Node Exporter and Deploy Web application on Instance 1

1. **Install Node Exporter**:

```
Wget
https://github.com/prometheus/node_exporter/releases/download/v1.8.1/no
de_exporter-1.8.1.linux-amd64.tar.gz tar xvfz node_exporter-1.8.1.linux-
amd64.tar.gz cd node_exporter-1.8.1.linux-amd64
./node_exporter &
```

2. **Install Nginx**: `sudo apt update` `sudo apt install nginx`

3. **Deploy Web Application(Java based Application)**

Git hub repo- https://github.com/jaiswaladi246/Boardgame.git

Install Java using this command.

```
sudo apt install openjdk-17-jre-headless -y
```

install Maven

```
sudo apt-get install software-properties-common
sudo apt-add-repository universe sudo apt-get
update sudo apt-get install maven
```

Build the Package `maven package`

Run the Application

```
cd target/
java -jar database_service_project-0.0.2.jar
```

Application runs on **Port 8080** -   <instance_ip>:8080

Phase 3: Install and Configure Prometheus, Blackbox Exporter, and Alertmanager on Instance 2

1. **Install Prometheus**:

```
wget
https://github.com/prometheus/prometheus/releases/download/v2.52.0/pro
metheus-2.52.0.linux-amd64.tar.gz

tar xvfz prometheus-2.52.0.linux-amd64.tar.gz cd

prometheus-2.52.0.linux-amd64

./prometheus --config.file=prometheus.yml &
```

2. **Install Blackbox Exporter**:

```
wget
https://github.com/prometheus/blackbox_exporter/releases/download/v0.25.
0/blackbox_exporter-0.25.0.linux-amd64.tar.gz    tar

xvfz  blackbox_exporter-0.25.0.linux-amd64.tar.gz  cd

blackbox_exporter-0.25.0.linux-amd64

./blackbox_exporter &
```

3. **Install Alertmanager**:

```
wget
https://github.com/prometheus/alertmanager/releases/download/v0.27.0/ale
rtmanager-0.27.0.linux-amd64.tar.gz

tar xvfz alertmanager-0.27.0.linux-amd64.tar.gz cd

alertmanager-0.27.0.linux-amd64

./alertmanager --config.file=alertmanager.yml &
```

Configuration Files

Prometheus Configuration (prometheus.yml)

Go inside the prometheus.yml file and add these configurations.

- **Global Configuration**:

```
global:
  scrape_interval:                    15s
evaluation_interval: 15s
```

- **Alertmanager Configuration**: `alerting:`

```
  alertmanagers:
- static_configs:
- targets: ['localhost:9093']
```

- **Scrape Configuration**:
  - **Prometheus**:

```
scrape_configs:
-      job_name: "prometheus"    static_configs:
-      targets: ["localhost:9090"]
```

  - **Node Exporter**: -

```
job_name: "node_exporter"
static_configs:
-      targets: ["<instance_ip>:9100"]
```

○ **Blackbox Exporter:**

```
- job_name: 'blackbox'
metrics_path: /probe  params:
  module: [http_2xx]
static_configs:
- targets:
- http://prometheus.io
- https://prometheus.io  -
http://<instance_ip>:8080/
relabel_configs:
- source_labels: [__address__]
target_label: __param_target  -
source_labels: [__param_target]
target_label: instance  -
target_label: __address__
replacement:<instance_ip>:9115
```

You should restart your Prometheus after completing all this configuration using this command . `pgrep Prometheus`

You will get some service id . example:- 3445 Kill

this service using command:

```
Kill 3445
```

Alert Rules Configuration (alert_rules.yml)

- **Alert Rules Group**: Create a new file inside the prometheus directory named **alert_rules.yml**

```
groups:
-       name: alert_rules
rules:
-       alert:
InstanceDown     expr:
up == 0     for: 1m
labels:
      severity: "critical"
annotations:
      summary: "Endpoint {{ $labels.instance }} down"
      description: "{{ $labels.instance }} of job {{ $labels.job }} has been down
for more than 1 minute."     - alert: WebsiteDown     expr: probe_success
== 0     for: 1m     labels:
      severity: critical     annotations:      description: The
website at {{ $labels.instance }} is down.
      summary: Website down
- alert: HostOutOfMemory
     expr: node_memory_MemAvailable / node_memory_MemTotal * 100 <
25
     for: 5m
```

```
    labels:
      severity: warning
annotations:
      summary: "Host out of memory (instance {{ $labels.instance }})"
      description: "Node memory is filling up (< 25% left)\n  VALUE = {{ $value
}}\n  LABELS: {{ $labels }}"      - alert: HostOutOfDiskSpace
    expr: (node_filesystem_avail{mountpoint="/"} * 100) /
node_filesystem_size{mountpoint="/"} < 50        for: 1s
labels:
      severity: warning
annotations:
      summary: "Host out of disk space (instance {{ $labels.instance }})"
description: "Disk is almost full (< 50% left)\n   VALUE = {{ $value }}\n
LABELS: {{ $labels }}"
-       alert: HostHighCpuLoad        expr: (sum by
(instance)
(irate(node_cpu{job="node_exporter_metrics",mode="idle"}[5m]))) > 80
for: 5m        labels:
      severity: warning
annotations:
      summary: "Host high CPU load (instance {{ $labels.instance }})"
description: "CPU load is > 80%\n  VALUE = {{ $value }}\n  LABELS: {{ $labels }}"
-       alert: ServiceUnavailable
    expr: up{job="node_exporter"} == 0
    for: 2m
labels:
     severity: critical
annotations:
     summary: "Service Unavailable (instance {{ $labels.instance }})"
description: "The service {{ $labels.job }} is not available\n  VALUE = {{
$value }}\n  LABELS: {{ $labels }}"      - alert: HighMemoryUsage
    expr: (node_memory_Active / node_memory_MemTotal) * 100 > 90
for: 10m     labels:
     severity: critical
annotations:
     summary: "High Memory Usage (instance {{ $labels.instance }})"
     description: "Memory usage is > 90%\n  VALUE = {{ $value }}\n  LABELS: {{
$labels }}"
   - alert: FileSystemFull
    expr: (node_filesystem_avail / node_filesystem_size) * 100 < 10
for: 5m     labels:
     severity: critical
annotations:
     summary: "File System Almost Full (instance {{ $labels.instance }})"
description: "File system has < 10% free space\n  VALUE = {{ $value }}\n
LABELS: {{ $labels }}"
```

Add alert_rules inside **prometheus.yml** - Uncomment the alert_rules.yml .

Create Gmail Authentication Password

To create an application-specific password for Gmail, you'll need to enable two-factor authentication (2FA) on your Google account and then generate an app password. Here are the steps:

1. **Enable Two-Factor Authentication (2FA):**
   - Log in to your Google account and navigate to Google Account Security.
   - Under the "Signing in to Google" section, find "2-Step Verification" and click on it.
   - Follow the prompts to set up 2FA. You might need to provide your phone number and verify it via a code sent by SMS.
2. **Generate an App Password:**
   - Once 2FA is enabled, go back to the Google Account Security page.
   - Search this on your browser :- https://myaccount.google.com/apppassowords
   - You might need to sign in again for security reasons.
   - On the "App passwords" page, click on the "Select app" dropdown and choose "Other (Custom name)".
   - Enter a name for the app password, such as "Prometheus Alertmanager", and click "Generate".
   - Google will provide you with a 16-character app password. Make sure to copy this password.
3. **Use the App Password in Your Configuration:**
   - Replace <your-app-password> in your alertmanager.yml configuration with the generated app password.

Alertmanager Configuration (alertmanager.yml)

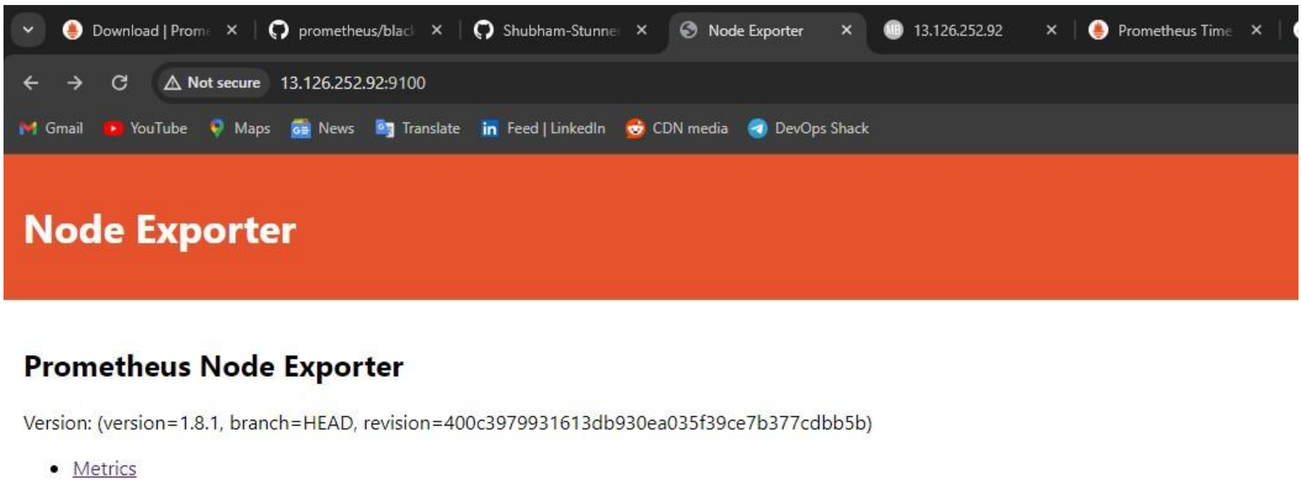
Routing Configuration: add these in the alertmanager.yml file

```
route:
  roup_by: ['alertname']
group_wait:          30s
group_interval:       5m
repeat_interval: 1h
  receiver: 'email-notifications'

receivers:
-   name: 'email-notifications'
email_configs:
-   to: jaiswaladi246@gmail.com
from: test@gmail.com        smarthost:
smtp.gmail.com:587
auth_username: your_email
auth_identity: your_email
        auth_password: "bdmq omqh vvkk zoqx"
send_resolved: true

inhibit_rules:   -
source_match:
      severity: 'critical'
target_match:
      severity: 'warning'
```

```
    equal: ['alertname', 'dev', 'instance']
```

Results

Node Exporter



13.126.252.92:9100

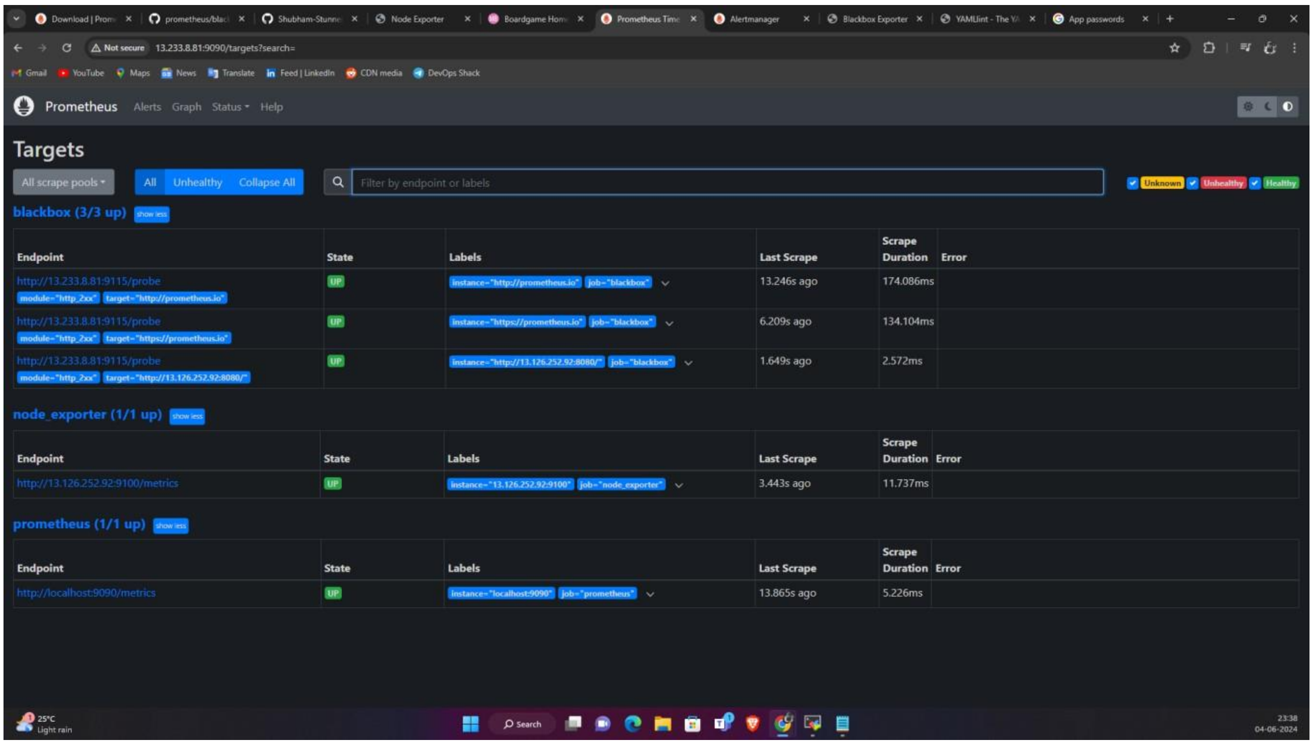
Node Exporter

Prometheus Node Exporter

Version: (version=1.8.1, branch=HEAD, revision=400c3979931613db930ea035f39ce7b377cdbb5b)

- Metrics

Prometheus



13.233.8.81:9090/targets?search=

Targets

blackbox (3/3 up)

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://13.233.8.81:9115/probe module="http_2xx" target="http://prometheus.io"	UP	instance="http://prometheus.io" job="blackbox"	13.246s ago	174.086ms	
http://13.233.8.81:9115/probe module="http_2xx" target="https://prometheus.io"	UP	instance="https://prometheus.io" job="blackbox"	6.209s ago	134.104ms	
http://13.233.8.81:9115/probe module="http_2xx" target="http://13.126.252.92:8080/"	UP	instance="http://13.126.252.92:8080/" job="blackbox"	1.649s ago	2.572ms	

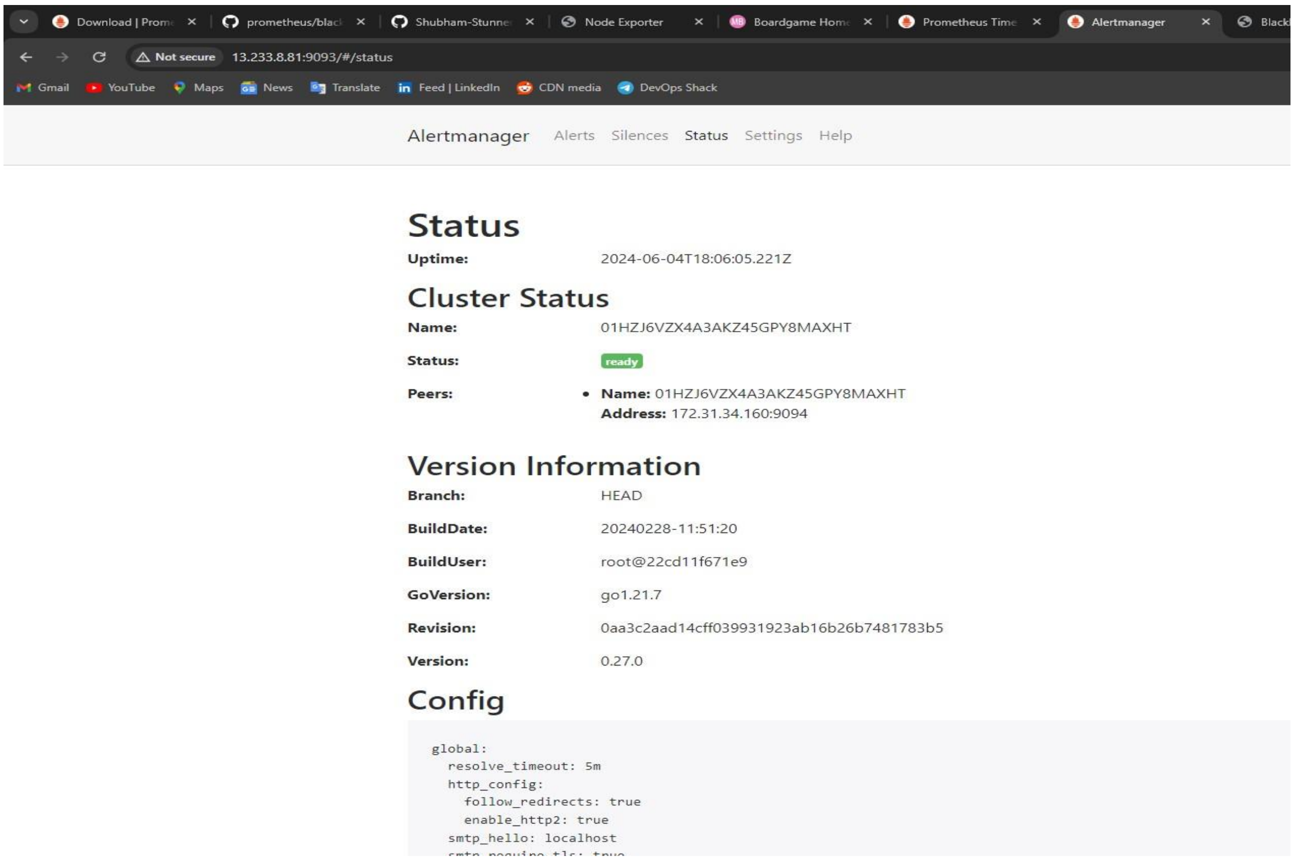
node_exporter (1/1 up)

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://13.126.252.92:9100/metrics	UP	instance="13.126.252.92:9100" job="node_exporter"	3.443s ago	11.737ms	

prometheus (1/1 up)

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://localhost:9090/metrics	UP	instance="localhost:9090" job="prometheus"	13.865s ago	5.226ms	

Alert manager



13.233.8.81:9093/#/status

Status

Uptime:	2024-06-04T18:06:05.221Z

Cluster Status

Name:	01HZJ6VZX4A3AKZ45GPY8MAXHT
Status:	ready
Peers:	• **Name:** 01HZJ6VZX4A3AKZ45GPY8MAXHT **Address:** 172.31.34.160:9094

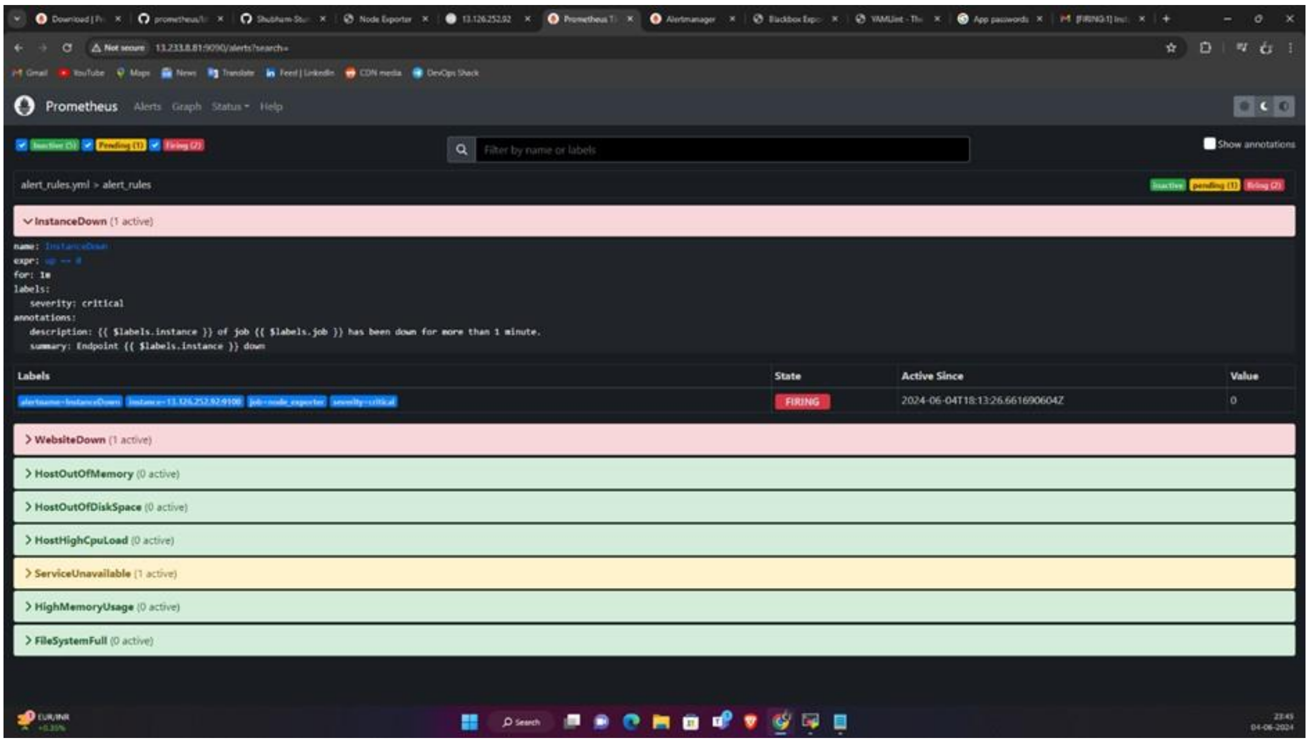
Version Information

Branch:	HEAD
BuildDate:	20240228-11:51:20
BuildUser:	root@22cd11f671e9
GoVersion:	go1.21.7
Revision:	0aa3c2aad14cff039931923ab16b26b7481783b5
Version:	0.27.0

Config

```
global:
  resolve_timeout: 5m
  http_config:
    follow_redirects: true
    enable_http2: true
  smtp_hello: localhost
```

Stop you web application & Node exporter to get an alert on alert manager.



13.233.8.81:9090/alerts?search=

alert_rules.yml > alert_rules

InstanceDown (1 active)

```
name: InstanceDown
expr: up == 0
for: 1m
labels:
  severity: critical
annotations:
  description: {{ $labels.instance }} of job {{ $labels.job }} has been down for more than 1 minute.
  summary: Endpoint {{ $labels.instance }} down
```

Labels	State	Active Since	Value
alertname=InstanceDown instance=13.126.252.92:9100 job=node_exporter severity=critical	FIRING	2024-06-04T18:13:26.661690604Z	0

WebsiteDown (1 active)

HostOutOfMemory (0 active)

HostOutOfDiskSpace (0 active)

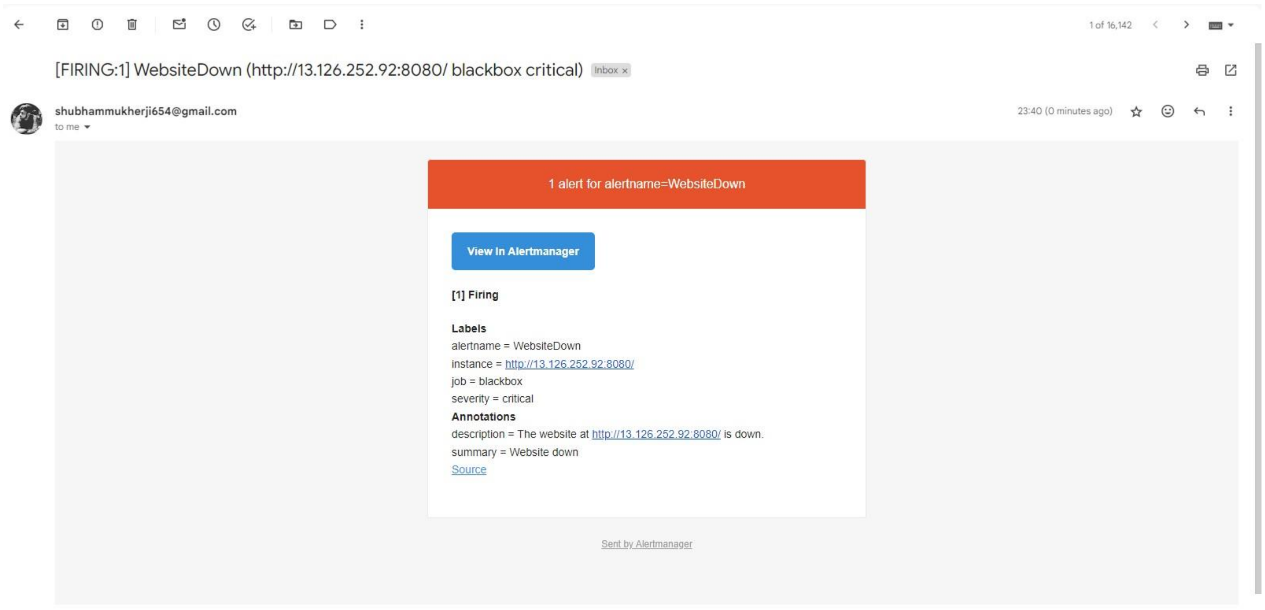
HostHighCpuLoad (0 active)

ServiceUnavailable (1 active)

HighMemoryUsage (0 active)

FileSystemFull (0 active)

Gmail Notification



[FIRING:1] WebsiteDown (http://13.126.252.92:8080/ blackbox critical)

shubhammukherji654@gmail.com

to me

23:40 (0 minutes ago)

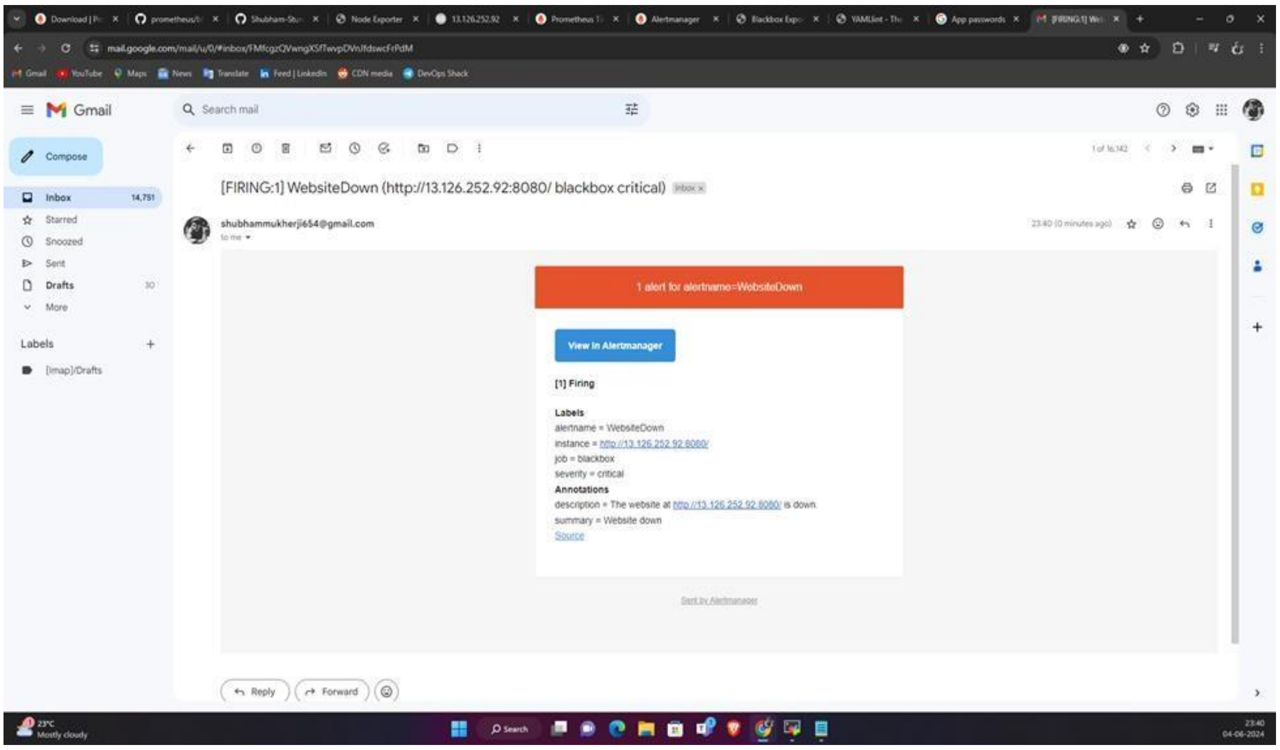
1 alert for alertname=WebsiteDown

View in Alertmanager

[1] Firing

Labels
alertname = WebsiteDown
instance = http://13.126.252.92:8080/
job = blackbox
severity = critical
Annotations
description = The website at http://13.126.252.92:8080/ is down.
summary = Website down
Source

Sent by Alertmanager



[FIRING:1] WebsiteDown (http://13.126.252.92:8080/ blackbox critical)

shubhammukherji654@gmail.com

1 alert for alertname=WebsiteDown

View in Alertmanager

[1] Firing

Labels
alertname = WebsiteDown
instance = http://13.126.252.92:8080/
job = blackbox
severity = critical
Annotations
description = The website at http://13.126.252.92:8080/ is down.
summary = Website down
Source

Sent by Alertmanager

Conclusion

In this project, we successfully implemented a robust monitoring solution using Prometheus and its ecosystem to ensure the reliability and performance of a web application hosted on AWS EC2 instances. By utilizing Node Exporter, Blackbox Exporter, and Alertmanager, we achieved a setup capable of collecting detailed metrics, monitoring endpoint availability, and managing alerts effectively.

Key achievements include:

- **Multi-Instance Setup**:

  ○ Instance 1: Hosts the web application, Node Exporter, and Nginx. ○ Instance 2: Hosts Prometheus, Blackbox Exporter, and Alertmanager.

- **Gmail Integration**: Configured Alertmanager to send email notifications via Gmail for timely alerts on critical issues.

This project highlights the importance of a well-planned monitoring strategy in maintaining the operational excellence of web-based services. Regular updates and adherence to best practices will ensure the monitoring solution remains effective and responsive to new challenges.